Reinforcement Learning
Mini-Project-2

Pielat Krystian

Sangineto Marina

Sauvenay Antoine

Sorbonne Université

2025

Contents

1 Introduction

In this project, we use the BBRL framework to study the effects of partial observability on the continuous-action version of the *LunarLander-v3* environment and *CartPoleContinuous-v1* environment with the TD3 algorithm.

To simulate partial observability, we implemented dedicated env wrappers. We investigate if extending the input to the agent's policy and critic with a memory of previous states helps to combat the challenges of partial observability. Additionally, we explore the impact of using action chunks(sequences of consecutive actions) rather than single-step actions, achieved through temporal extension wrappers.

The study focuses on a single performance metric: the mean reward. The goal of the trained algorithms is to maximize it. Both environments have different arbitrary reward calculation rules. In *LunarLander-v3*, the agent controls a lander using continuous main and side thrusters to land softly on a designated pad. Reward increases as the lander approaches the pad, maintains upright orientation, and lands gently, while penalties apply for speed, tilt, or fuel use. Crashes give large negative rewards, and successful landings give large bonuses. In *CartPoleContinuous-v1*, the agent applies continuous horizontal force to keep a pole balanced upright on a moving cart. The reward is $+1$ for every timestep the pole remains balanced within bounds, ending when the pole falls or the cart moves too far.

2 Wrappers

The `FeatureFilterWrapper` removes a specific feature from the returned observation during calls to the `reset()` and `step(action)` functions. The feature to be removed is specified as an index when constructing the wrapper object.

For example, to filter out the x and y velocities of the lander in the LunarLander-v3 environment, the wrapper can be applied multiple times as follows:

```
env = FeatureFilterWrapper(FeatureFilterWrapper(inner_env, X), Y)
```

where `inner_env` refers to the environment, and `X` and `Y` represent the indices of the features to be filtered out.

2.1 Exercise 1: Implementing the `FeatureFilterWrapper` Class

The purpose of the `FeatureFilterWrapper` is to simulate partial observability by removing selected features from the observation vector. This allows evaluating TD3's robustness with incomplete state information. The wrapper modifies the `reset()` and `step(action)` functions to filter out specified features before returning observations.

2.2 Exercise 2: Implementing the `ObsTimeExtensionWrapper` Class

The `ObsTimeExtensionWrapper` provides the agent with short-term memory by concatenating the current observation with the previous observation. This temporal extension allows the agent to infer velocity and acceleration information that may have been removed by partial observability. The implementation maintains a `previous_obs` buffer initialized with zeros during `reset()` and updated after each `step()`.

2.3 Exercise 3: Implementing the `ActionTimeExtensionWrapper` Class

The `ActionTimeExtensionWrapper` modifies the action space by requiring the agent to output a sequence of M consecutive actions at each decision step, but only the first action is executed by the environment.

3 Experimental Study

In this section presents the experimental study comparing TD3 performance under various observability conditions and temporal extension configurations.

3.1 Architectural Choices

For this exercise, the TD3 (Twin Delayed Deep Deterministic Policy Gradient) algorithm was used in the continuous version of the *LunarLander-v3* environment, as required. We decided to additionally run the experiments on the *CartPoleContinuous-v1* environment and tune new set of hyperparameters for it. TD3 is characterized by its robustness and stability in continuous control problems, relying on twin critics, delayed policy updates, and target smoothing to reduce overestimation bias.

For exploration, we employed Ornstein–Uhlenbeck noise using the `AddOUNoise` agent. This temporally correlated noise model is particularly suitable for continuous control tasks, as it produces smoother exploration trajectories compared to uncorrelated Gaussian noise.

We initially attempted to optimize the model hyperparameters in each environment using *Optuna*. However, due to the high computational cost of the training process, we decided to resign from full quantitative hyperparameter optimization and instead tuned the parameters empirically by testing multiple configurations and comparing their performance in TensorBoard.

3.2 Experimental Setup

To systematically evaluate the impact of partial observability and temporal extensions, we designed four experimental configurations:

1. **Baseline / Full observability**: Full observability with the original 8-dimensional observation space of LunarLander-v3. This serves as the reference performance level.

2. **Partial Observability**: Removed the horizontal and vertical velocities ($\dot{x}$ and $\dot{y}$) using two nested `FeatureFilterWrapper` instances, reducing the observation space.

3. **Observation Memory**: Applied `ObsTimeExtensionWrapper` to the baseline environment, expanding the observation space by concatenating current and previous observations.

4. **Action Chunking**: Applied `ActionTimeExtensionWrapper` with sequence length $M = 3$, expanding the action space from 2 to 6 dimensions. Only the first action in each sequence is executed, promoting temporal consistency in the policy.

Apart from these main configurations, we also tested several combined setups to evaluate how *observation memory* and *action chunking* could mitigate the loss of information caused by partial observability.

Each configuration was trained for 1000 episodes using 5 random seeds (1-5) to ensure statistical robustness. All experiments used identical TD3 hyperparameters (different set for LunarLander and different for CartPole, they can be found here).

3.3 Results and Analysis

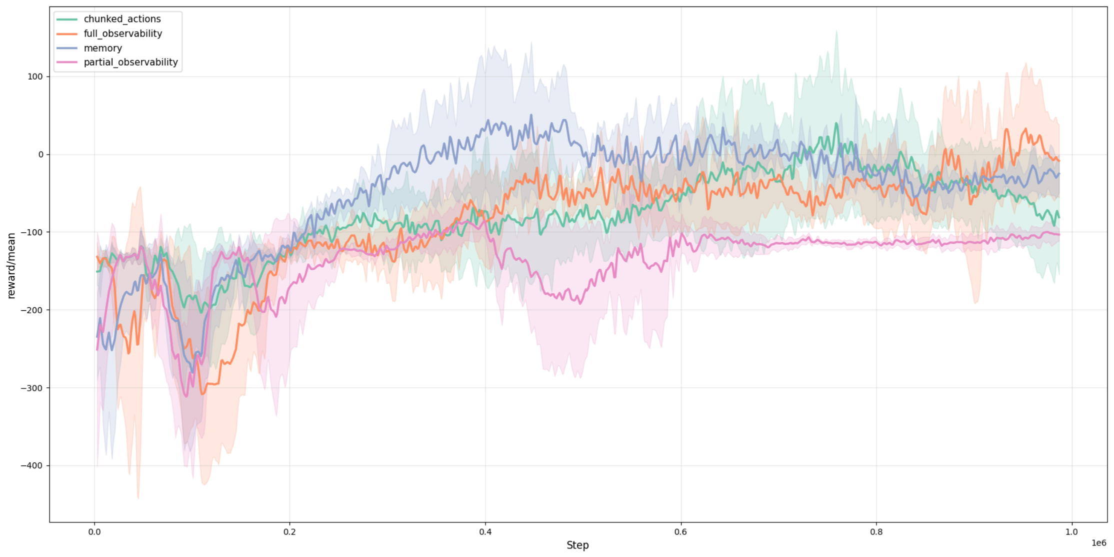

Figure 1: Results without combining wrappers.

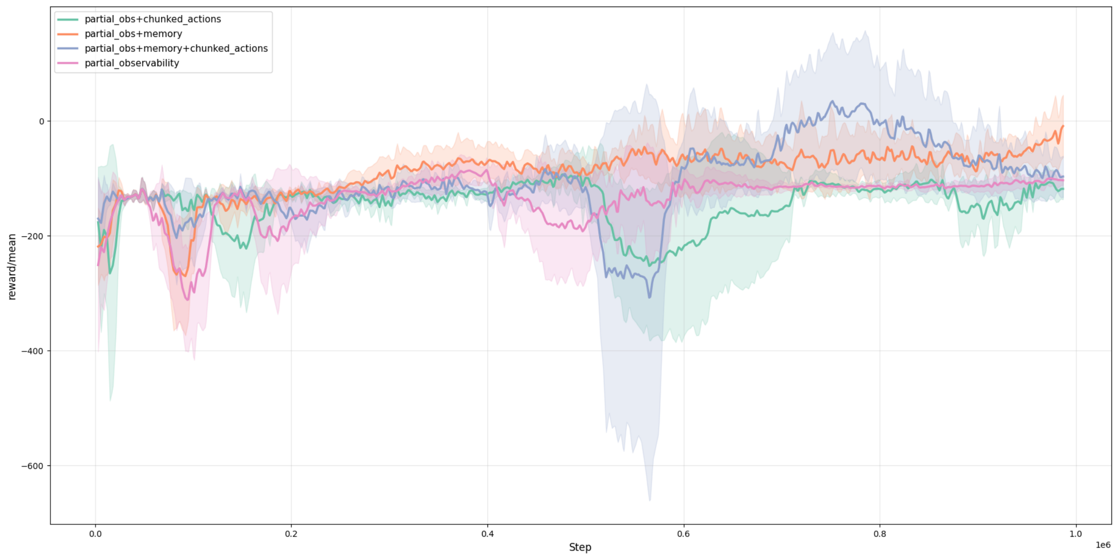

Figure 2: Results of partial observability configurations combined with other wrappers.

3.3.1 General Results

Based on figure 1 all algorithms demonstrated stable learning and consistent improvement. The results would be more insightful if they were ran on more than 5 seeds per configuration (more precise confidence intervals) and with more training steps, however our study was limited by computational resources.

The performance of the baseline configuration with full observability was approximately in the middle between the other configurations, which aligns with intuition, given it was neither hindered by removing information about the environment, nor improved by providing additional information like memory.

3.3.2 Partial Observability

When horizontal and vertical velocities were removed (*partial_ observability* configuration), we see on figure 1 that performance degradation was observable. The agent struggled more during training, slower convergence and plateauing around lower average reward than other algorithms. This confirms that velocity information is critical for optimal landing control - without it, the agent must rely on position changes across timesteps to estimate velocities, which introduces noise and delays in control responses.

3.3.3 Results of wrappers

The `ObsTimeExtensionWrapper` (*memory* configuration) provided substantial improvements over other configurations, performing the best out of all the algorithms, even better than the baseline *full_ observability*.

The `ActionTimeExtensionWrapper` (*chunked_ actions*) showed mixed results, mostly similar to the baseline. Based on the data it doesn't seem to be providing much additional information for the model and doesn't improve the performance of *full_ observability* configuration.

For some seeds, action chunking eventually led to smoother control policies with better fuel efficiency, but convergence was slower and less reliable across seeds. This suggests that action temporal extension may be more beneficial in tasks requiring precise trajectory planning rather than reactive control like lunar landing.

3.3.4 Influence of memory and action chunking on mitigating partial observability

Based on figure 2 it seems that combinations involving memory managed to mitigate to some extent the loss of information caused by removing the information about lunar lander's velocity in x and y axes. Using chunked actions doesn't seem to be improving the performance, or possibly even worsens it.

3.3.5 Statistical Significance

We noticed that the results have relatively high variance. Based on the data we collected it's difficult to conclusively say which algorithms perform best, due to too little seeds and training steps. The plots contains 95% confidence intervals to give more insight into the variability in the data (the series are means of configurations across seeds).

4 Conclusion

This study systematically investigated the effects of partial observability on TD3 learning in the LunarLander-v3 environment and evaluated two temporal extension strategies for mitigating performance degradation.